APPARATUS AND METHODS FOR MANAGING CACHES ON A MOBILE DEVICE

FIELD OF THE INVENTION

This invention relates to apparatus and methods for managing caches. In particular, this invention relates to apparatus and methods for managing caches on a mobile device.

BACKGROUND OF THE INVENTION

Generally, wireless/mobile devices include a user interface, such as a micro-browser, pre-installed on a wireless/mobile device and a set of fixed applications and hierarchical menus for Internet access. Using the micro-browser, a user browses the Internet using the fixed menus or by manually entering specific uniform resource locators (URLs).

Most wireless/mobile devices have inadequate processing capability for retrieving information, such as applications or data, and very limited memory space for caching such information. Thus, downloading applications or data from the Internet onto a mobile device may be very slow and sometimes unsuccessful. One possible solution to circumvent the need to repeatedly download from the Internet is to cache applications and data on the mobile device. Because mobile devices have very limited memory space, an intelligent caching of the most likely to be called applications or data is necessary to optimize this solution. Moreover, due to size and other limitations in mobile devices, efficient management of the cache space is necessary to ensure that application and data stored in the cache are up-to-date and useful to users.

Thus, it is desirable to provide apparatus and methods for managing caches on a mobile device.

SUMMARY OF THE INVENTION

An exemplary method for managing a cache on a mobile device comprises the steps of receiving a call for loading a set of files ("the original set of files"), the original set of files including an application or data, searching a database for a matching record to the original set of files, determining if the original set of files is out-of-date or if a scheduled update is overdue based on the matching record, updating the original set of files if it is out-of-date, performing a status check or update if the scheduled update is overdue, and loading the original set of files if it is not out-of-date and the scheduled update is not overdue. In one embodiment, the exemplary method further comprises the step of loading an updated set of files after an update or status check is complete.

In an exemplary embodiment, updating the original set of files includes the steps of opening a communications session with a gateway or a remote server, sending an update request to the gateway or the remote server, receiving an update response, the response including at least one difference file for updating the original set of files, closing the communications session, and updating a local file system and the database based on the update response.

In one embodiment, the local file system and the database are updated by loading the at least one difference file into a random access memory, loading a copy of the original set of files from the local file system into the random access memory, applying the at least one difference file on the original set of files in the random access memory to obtain an updated set of files, copying the updated set of files into the local file system, removing the original set of files from the local file system, updating the database based on the updated set of files, and removing the at least one difference file and the original set of files from the random access memory.

In another embodiment, the step of applying the at least one difference file to the original set of files in the random access memory includes the steps of parsing the at least one difference file to determine whether to add, modify, or delete a file in the original set of files and updating the local file system and the database based on the parsing.

In another exemplary embodiment, the exemplary method further comprises the steps of parsing the update response for a broadcast message, accessing and updating the database based on the broadcast message, sending a broadcast response to

the gateway or the remote server. In one embodiment, the accessing and updating steps include the step of selectively marking at least one set of files as out-of-date.

In yet another exemplary embodiment, the step of performing a status check or update includes the steps of opening a communications session with a gateway or a remote server, sending a status check or update request to the gateway or the remote server, receiving a status check or update response from the gateway or the remote server, closing the communications session, and updating a local file system and a database if at least one difference file is included in the status check or update response.

In one embodiment, the local file system and the database are updated by loading the at least one difference file into a random access memory, loading a copy of the original set of files from the local file system into the random access memory, applying the at least one difference file on the original set of files in the random access memory to obtain an updated set of files, copying the updated set of files into the local file system, removing the original set of files from the local files system, updating the database based on the updated set of files, and removing the at least one difference file and the original set of files from the random access memory. In another embodiment, the step of applying the at least one difference file on the original set of files in the random access memory includes the steps of parsing the at least one difference file to determine whether to add, modify, or delete a file in the original set of files and updating the local file system and the database based on the parsing.

In yet another embodiment, the step of performing a status check or update further comprises the steps of parsing the status check or update response for a broadcast message, selectively marking at least one set of files as out-of-date based on the broadcast message, and sending a broadcast response to the gateway or the remote server.

An exemplary computer program product for managing a cache on a mobile device comprises logic code for receiving a call for loading a set of files ("the original set of files"), the original set of files including an application or data, logic code for searching a database for a matching record to the original set of files, logic code for determining if the original set of files is out-of-date or if a scheduled update is overdue based on the matching record, logic code for updating the original set of files if it is out-of-date, logic code for performing a status check or update if the scheduled update is overdue, and logic code for loading the original set of files if it is not out-of-date and

the scheduled update is not overdue. In one embodiment, the exemplary computer program product further comprises logic code for loading an updated set of files after an update or status check is complete.

In an exemplary embodiment, the logic code for updating the original set of files includes logic code for opening a communications session with a gateway or a remote server, logic code for sending an update request to the gateway or the remote server, logic code for receiving an update response, the response including at least one difference file for updating the original set of files, logic code for closing the communications session, and logic code for updating a local file system and the database based on the update response.

In one embodiment, the logic code for updating a local file system and the database includes logic code for loading the at least one difference file into a random access memory, logic code for loading a copy of the original set of files into the random access memory, logic code for applying the at least one difference file on the original set of files in the random access memory to obtain an updated set of files, logic code for copying the updated set of files into the local file system, logic code for removing the original set of files from the local file system, logic code for updating the database based on the updated set of files, and logic code for removing the at least one difference file and the original set of files from the random access memory. In another embodiment, the logic code for applying the at least one difference file to the original set of files in the random access memory includes logic code for parsing the at least one difference file to determine whether to add, modify, or delete a file in the original set of files and logic code for updating the local file system and the database based on the parsing.

In another exemplary embodiment, the exemplary computer program product further comprises logic code for parsing the update response for a broadcast message, logic code for accessing and updating the database based on the broadcast message, logic code for sending a broadcast response to the gateway or the remote server. In one embodiment, the logic code for accessing and updating includes logic code for selectively marking at least one set of files as out-of-date.

In yet another exemplary embodiment, the logic code for performing a status check or update includes logic code for opening a communications session with a gateway or a remote server, logic code for sending a status check or update request to the gateway or the remote server, logic code for receiving a status check or update

response from the gateway or the remote server, logic code for closing the communications session, and logic code for updating a local file system and a database if at least one difference file is included in the status check or update response.

In one embodiment, the logic code for updating a local file system and a database includes logic code for loading the at least one difference file into a random access memory, logic code for loading a copy of the original set of files into the random access memory, logic code for applying the at least one difference file on the original set of files in the random access memory to obtain an updated set of files, logic code for copying the updated set of files into the local file system, logic code for removing the original set of files from the local file system, logic code for updating the database based on the updated set of files, and logic code for removing the at least one difference file and the original set of files from the random access memory. In another embodiment, the logic code for applying the at least one difference file to the original set of files in the random access memory includes logic code for parsing the at least one difference file to determine whether to add, modify, or delete a file in the original set of files and logic code for updating the local file system and the database based on the parsing.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 schematically illustrates an exemplary prior art system.

FIGURE 2 schematically illustrates an exemplary mobile device in accordance with an embodiment of the invention.

FIGURE 3 schematically illustrates an exemplary two level transaction support process in accordance with an embodiment of the invention.

FIGURE 4 illustrates an exemplary application identification table in accordance with an embodiment of the invention.

FIGURE 5 illustrates an exemplary data identification table in accordance with an embodiment of the invention.

FIGURE 6 illustrates an exemplary compression methods table in accordance with an embodiment of the invention.

FIGURE 7 illustrates an exemplary application download table in accordance with an embodiment of the invention.

FIGURE 8 illustrates an exemplary data download table in accordance with an embodiment of the invention.

FIGURE 9 illustrates an exemplary application storage table in accordance with an embodiment of the invention.

FIGURE 10 illustrates an exemplary data storage table in accordance with an embodiment of the invention.

FIGURE 11 illustrates an exemplary application execution table in accordance with an embodiment of the invention.

FIGURE 12 illustrates an exemplary data access table in accordance with an embodiment of the invention.

FIGURE 13 illustrates an exemplary application cache change table in accordance with an embodiment of the invention.

FIGURE 14 illustrates an exemplary data cache change table in accordance with an embodiment of the invention.

FIGURE 15 illustrates an exemplary configuration table in accordance with an embodiment of the invention.

FIGURE 16 illustrates an exemplary process in accordance with an embodiment of the invention.

FIGURE 17 illustrates another exemplary process in accordance with an embodiment of the invention.

FIGURE 18 illustrates another exemplary process in accordance with an embodiment of the invention.

FIGURE 19 illustrates another exemplary process in accordance with an embodiment of the invention.

FIGURE 20 illustrates another exemplary process in accordance with an embodiment of the invention.

FIGURE 21 schematically illustrates exemplary smart connectivity protocol state machines in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 illustrates an exemplary prior art system 100. The system 100 includes multiple servers connected to multiple gateways that service multiple mobile devices. For ease of explanation, only a representative number of servers, gateways, and mobile devices are shown in Figure 1. The system 100 includes servers 102A-102C, gateways 108A-108B, and mobile devices 110A-110C.

Figure 2 schematically illustrates an exemplary mobile device 110 in accordance with an embodiment of the invention. The mobile device 110 includes a communications interface 202 for communicating with a network, a microprocessor 204, a user interface 206, and a memory 208. In an exemplary embodiment, the user interface includes a user input device (e.g., keyboard) and an output device (e.g., screen). The memory 208 includes an operating system 210, a micro-browser application 212, a user operation history tracking module 214 for tracking user operation history, a smart connectivity module 216, a mobile database 218, a local file system 226 (e.g., built-in flash memory), a download manager 228, a cache engine 230, a smart connectivity protocol 232, a communications transport protocol module 234 for adapting to different transport protocols in the network, and a random access memory (RAM) 236. In an exemplary embodiment, the mobile database 218 includes a set of application tables 220, a set of data tables 222, and a set of other tables 224 for tracking user operation and cache storage information.

In an exemplary embodiment, the micro-browser application 212 provides a graphical user interface to a user. In one embodiment, a list of applications may be presented via the micro-browser application 212 to the user for receiving user selection. Each item in the list of applications includes a uniform resource locator (URL) and a brief description of the application. For example, the brief description includes a function description, product promotion, or URLs to other related web pages. In an exemplary embodiment, the user can select an application by browsing the list and highlighting the application or by entering an application number. When an application is selected, it is either loaded from the local file system 226, from the gateway 108, or from a remote server 102. The application selection information is tracked by the user operation history tracking module 214 and recorded in the application tables 220 and data tables 222 in the mobile database 218.

The smart connectivity module 216 determines whether an application or data requested for execution or access is already stored in the local file system 226 and sends a request to the gateway 108 or a remote server 102 via the download manager 228 to download the requested application or data if it is not stored in the local file system 226. The smart connectivity module 216 calls the cache engine 230 to intelligently determine (based on a calculated cache benefit index) whether a downloaded application/data should be cached, and if so, whether there is enough space to do so. Detailed description regarding the cache benefit index (CBI) and

methods to intelligently cache applications and data on a mobile device is provided in a pending application entitled "Apparatus and Methods for Intelligently Caching Applications and Data on a Mobile Device," bearing application serial No. __________, filed on __________. This pending application is hereby incorporated for all purposes. Additionally, the smart connectivity module 216 maintains meta information (i.e., synchronization version and app/data identification) for all cached application/data in the mobile database 218 in one or more of the tables 220-224.

When an application/data is downloaded, all files belonging to that application/data, including the executable files, configuration files, property files, online help files, etc., are downloaded as a bundle. Similarly, when an application/data is cached, all files belonging to that application/data are cached in the local file system 226.

In an exemplary embodiment, if the gateway 108 and the server 102 are compatible (3i) gateway and (3i) server, respectively, communications between the mobile device 110 and the gateway 108 or the server 102 are based on the smart connectivity protocol 232 that is stacked on top of the communication transport and protocol 234 (e.g., wireless application protocol (WAP), TCP/IP, HTTP, infra-red data association (IrDA), or Bluetooth). If the gateway 108 and the server 102 are not compatible (non-3i), then communications between the mobile device 110 and such gateway and server are based only on the communication transport and protocol 234. Additional description relating to the 3i gateway and the 3i server is disclosed in co-pending applications entitled "Apparatus and Methods for Intelligently Caching Applications and Data on a Gateway," bearing serial number __________, filed on __________ and "Apparatus and Methods for Managing Caches on a Gateway," bearing serial number __________, filed on __________. These applications are hereby incorporated by reference for all purposes.

Figure 3 illustrates an exemplary transaction and sub-transaction management in accordance with an embodiment of the invention. During each application/data downloading or application/data updating, the smart connectivity module 216 maintains the consistency and integrity among database operations and application/data cache space management. A transaction corresponding to an application/data update or status check is created after the smart connectivity module 216 initiates the update or status check request on the mobile device 110. The transaction is committed when the smart connectivity module 216 succeeds in the

update or status check processes; otherwise, if the smart connectivity module 216 fails in the processes, the transaction is rolled back to its original state. In an exemplary embodiment, during a transaction processing, the smart connectivity module 216 may also create several sub-transactions within the transaction for various database operations. For example, the sub-transactions include an application or data cache space management transaction and communication transactions with the gateway 108 or a remote server 102. Sub-transactions become fully committed when the initial transaction becomes committed.

In an exemplary embodiment, the mobile database 218 includes a number of tables 220-224. Each table is designed to maintain a type of logical information. The smart connectivity module 216 updates the mobile database 218 and the local file system 226 in accordance with each operation performed. For example, if a requested application or data is already preloaded or cached, the smart connectivity module 216 updates the corresponding application execution table (see Figure 11 below) or data access table (see Figure 12 below). If a requested application or data is not already cached, the smart connectivity module 216 calls the download manager 228 to download the application or data. Next, the smart connectivity module 216 updates the application download table (see Figure 7 below) or the data download table (see Figure 8 below). The smart connectivity module 216 then calls the cache engine 230 to determine whether to cache the downloaded application. If so, the application or data is cached, and the smart connectivity module 216 updates the application storage table (see Figure 9 below) or the data storage table (see Figure 10 below) and the application cache change table (see Figure 13 below) or the data cache change table (see Figure 14 below).

The mobile database 218 is managed in the mobile device 110 by either a third-party (commercially available) database management system or a built-in micro database management system in the mobile operation system 210. In an exemplary embodiment, twelve tables are maintained in the mobile database 218. Exemplary tables are illustrated in Figures 4-15 below.

Figure 4 illustrates an exemplary application identification table. The purpose of this table is to associate each application uniform resource locator (URL) to a unique identifier.

Figure 5 illustrates an exemplary data identification table. The purpose of this table is to associate each data URL to a unique identifier.

Figure 6 illustrates an exemplary compression methods table. The purpose of this table is to associate each data compression method name to a unique identifier.

Figure 7 illustrates an exemplary application download table. The purpose of this table is to track the download histories of all applications downloaded by the mobile device 110.

Figure 8 illustrates an exemplary data download table. The purpose of this table is to track the download histories of all data downloaded by the mobile device 110.

Figure 9 illustrates an exemplary application storage table. The purpose of this table is to maintain the meta information associated with all cached applications in the mobile device 110.

Figure 10 illustrates an exemplary data storage table. The purpose of this table is to maintain the meta information associated with all cached data in the mobile device 110.

Figure 11 illustrates an exemplary application execution table. The purpose of this table is to track the execution histories of all downloaded applications at the mobile device 110.

Figure 12 illustrates an exemplary data access table. The purpose of this table is to track the access histories of all downloaded data at the mobile device 110.

Figure 13 illustrates an exemplary application cache change table. The purpose of this table is to maintain a list of application URLs that have been swapped in or out of the local file system 226 until the information is transferred to the gateway 108.

Figure 14 illustrates an exemplary data cache change table. The purpose of this table is to maintain a list of data URLs that have been swapped in or out of the local file system 226 until the information is transferred to the gateway 108.

Figure 15 illustrates an exemplary configuration table. The purpose of this table is to set and maintain a set of configuration parameters that control the behavior of the mobile device 110.

Figure 16 illustrates an exemplary application loading process in accordance with an embodiment of the invention. At step 1602, a call for an application loading is received. Next, the application storage table (see Figure 9) is searched for a record that matches the called application (step 1604). If there is no matching record (step 1606), the process ends (i.e., the application will be downloaded from the gateway 108 or the server 102). If there is a matching record (step 1606), whether the "flagset"

field of the matching record indicates that the corresponding application is out-of-date is determined (step 1608). If the flagset field indicates that the application is out-of-date, the process continues in Figure 17. Otherwise, whether the "nextRel" field in the matching record has expired is determined (step 1610). In an exemplary embodiment, the nextRel field has expired if it indicates a point in time older than the current time. If the nextRel has expired, the process continues in Figure 18. Otherwise, whether the nextRel field is equal to zero is determined (step 1612). If the nextRel field is equal to zero, the process continues in Figure 18. Otherwise, the application is up-to-date and is loaded from the local file system (step 1614).

Figure 17 illustrates an exemplary update process in accordance with an embodiment of the invention. At step 1702, a request to open or reuse a communications session is sent to the gateway 108 or the server 102. Next, a response is received from the gateway 108 or the server 102 (step 1704). An application update request is sent to the gateway 108 or the server 102 (step 1706). In an exemplary embodiment, the application update request includes the following information: the application URL, the number of files in the application, and the name and version of each file in the application. A response to the update request is received from the gateway 108 or the remote server 102 (step 1708). In an exemplary embodiment, the response includes at least one difference file for updating the application differentially. The response is parsed to determine whether a broadcast is piggybacked (step 1710). If so, the application storage table (see Figure 9) is accessed and updated (step 1712). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* field of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, a broadcast response is sent back to the gateway 108 or the remote server 102 (step 1714) and the process continues at step 1716. Referring back to step 1710, if no broadcast information is piggybacked, the process continues at step 1716.

At step 1716, a close session request is sent to the gateway 108 or the server 102 and the communication is disconnected. The local file system 226 and the

application storage table (see Figure 9) are updated (step 1718). In an exemplary embodiment, the local files system 226 is updated by applying the received at least one difference file to the corresponding application cached in the local file system 226 to obtain an updated application. In an exemplary embodiment, the following fields in the application storage table are updated: number of files, file names, versions, next application release schedule, language, flagset, nUpdate, updateRate, and CBI. In one embodiment, the new nUpdate = old nUpdate +1, the new updatRate = [(old updateRate x old nUpdate) + (diffSize x 100/appSize)]/new nUpdate, and the new CBI = old CBI - diffSize, where diffSize is the size difference between the new and old application versions. In an exemplary embodiment, some or all originally cached applications may have to be removed from the local file system 226 to create space for storing the updated application. In such a case, records corresponding to the removed applications are also removed from the application storage table (see Figure 9) and the application cache change table (see Figure 13) is updated to reflect the removal (step 1720). Next, the application execution table (see Figure 11) is updated (step 1722). An updated record contains the time stamp, version information, and per-execution CBI related to the current execution of the updated application.

In an exemplary embodiment, application/data update is performed when the mobile device 110 is not in use by the user (e.g., user is making a call or executing any application). In addition, an on-going updating process is interrupted and terminated when an incoming call is received or if the user wishes to terminate the updating process. In one embodiment, an interrupted updating process can be resumed later at the point/state of interruption. In another embodiment, the updating process is executed in a transactional manner. That is, an updating process is either finished completely or the application/data being updated returns to its original state when an attempted update is interrupted.

Figure 18 illustrates an exemplary application status check or update process in accordance with an embodiment of the invention. At step 1802, a request to open or reuse a communications session is sent to the gateway 108 or the server 102. Next, a response is received from the gateway 108 or the server 102 (step 1804). An application status check or update request is sent to the gateway 108 or the server 102 (step 1806). In an exemplary embodiment, the application status check or update request includes the following information: the application URL, the application version, the number of files in the application, and the name and version of each file in

the application. A response to the application status check or update request is received from the gateway 108 or the server 102 (step 1808). In an exemplary embodiment, the response includes a current version status of the application or at least one difference file for updating the application. The response is parsed to determine whether a broadcast message is piggybacked (step 1810). If a broadcast message is piggybacked (step 1812), the application storage table (see Figure 9) is accessed and updated (step 1814). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* field of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, a broadcast response is sent back to the gateway 108 or the remote server 102 (step 1816) and the process continues at step 1818. Referring back to step 1812, if no broadcast information is piggybacked, the process continues at step 1818.

At step 1818, a close session request is sent to the gateway 108 or the server 102 and the communication is disconnected. Next, whether an update response is received from the gateway 108 or the server 102 is determined (step 1820). If not, the process ends. If an update response is received, the local file system 226 and the application storage table (see Figure 9) are updated (step 1822). In an exemplary embodiment, the local file system 226 is updated by applying the received at least one difference file to the corresponding application cached in the local file system to obtain an updated application. In an exemplary embodiment, the following fields in the application storage table are updated: number of files, file names, versions, next application release schedule, language, flagset, nUpdate, updateRate, and CBI. In one embodiment, the nUpdate, updatRate, and CBI are updated as described above in Figure 17. In an exemplary embodiment, some or all originally cached applications may have to be removed from the local file system 226 to create space for storing the updated application. In such a case, records corresponding to the removed applications are removed from the application storage table (see Figure 9) and the application cache change table (see Figure 13) is updated to reflect the removal (step 1824). Next, the application execution table (see Figure 11) is updated (step 1826). An updated record

contains the time stamp, version info, and pre-execution CBI related to the current execution of the updated application.

An application or data typically comprises a set of files. When an application or data is updated, one or more of a corresponding set of files is updated (i.e., added, modified, or removed). In an exemplary embodiment, at least one difference file is created by the gateway 108 or the server 102 that represents the difference between the old version and the new version of the application to be updated. A difference file provides information regarding a file to be added to an original set of files, a file in an original set of files that should be modified, or a file in an original set of files that should be deleted. For example, to add a file, a difference file includes the file's name, a 16-byte version information, contents of the new file, and the size of the new file in bytes. To delete a file, a difference file includes the name of the file to be deleted. To modify a file, a difference file includes a description of the difference between the modified file and the original file or the contents of the modified file, whichever is smaller.

Figure 19 illustrates an exemplary process to differentially update a file in an application in accordance with an embodiment of the invention. At step 1902, a difference file is received. The received difference file is parsed (step 1904). Next, whether a file is to be added to an original set of files is determined (step 1906). If so, the file is added to the original set of files (step 1908). If no file is to be added, whether a file is to be deleted is determined (step 1910). If so, the appropriate file is deleted from the original set of files (step 1912). If no file is to be deleted, by default, at least a file in the original set of files should be modified. Thus, whether the contents of the file to be modified is included in the difference file is determined (step 1914). If the contents of the file to be modified is included, that file in the original set of files is replaced with a modified file in the difference file (step 1916). Otherwise, that file in the original set of files is modified in accordance with instructions in the difference file (step 1918).

Figure 20 illustrates an exemplary application update process that reduces the probability of application corruption in accordance with an embodiment of the invention. At step 2002, at least one difference file is received from the gateway 108 or the server 102. The at least one difference file is placed into the RAM 236 of the mobile device 110 (step 2004). A copy of a set of files of an application to be updated is loaded from the local file system 226 into the RAM 236 (step 2006). The at least one difference file is applied to the set of files in the RAM 236 as described in Figure

19 above to obtain an updated set of files (step 2008). Next, a copy of the updated set of files is saved in the local file system 226 (step 2010). In an exemplary embodiment, the original set of files is removed or overwritten in the process of copying the updated set of files (step 2012). Next, the current application update transaction is reflected in the mobile database 218 (step 2014). For example, appropriate fields in the application cache change table and the application storage table are updated. The at least one difference file and the original set of files are removed from the RAM 236 (step 2016). Next, depending on the application type and/or the operation system features, whether the updated application should be executed is determined (step 2018). If so, the updated application may remain in the RAM 236 until it is executed (step 2020). If the updated application is not to be executed or after an execution, the updated application is removed from the RAM 236 (step 2022).

In an exemplary embodiment, to further ensure data integrity, the smart connectivity module 216 scans the RAM 236 (or the application cache space) and the mobile database 218 after each power up of the mobile device 110 to identify any data inconsistency among them. And, if necessary, the smart connectivity module 216 repairs the mobile database 218 to correct any data inconsistency.

Although Figures 16-20 illustrate exemplary processes to process applications, these processes similarly apply to data.

The smart connectivity protocol (SCP) is a protocol used for application/data management between the mobile device 110 and the gateway 108 or between the mobile device 110 and a remote server 102. Figure 21 illustrates exemplary state machines of the SCP in accordance with an embodiment of the invention. Generally, when the SCP is in an Idle state, no communication session is created and, thus, no communication activity is taking place. When the SCP is in an Open state, a communication session is created; the system may be for communication requests from a client. When the SCP is in a Download state, a download request is sent or a download response is prepared. When the SCP is in an Update state, an update request is sent or an update response is prepared. When the SCP is in an Initialize state, an initialization request is sent or an initialization is prepared. When the SCP is in a Register state, cache changes are piggybacked or an acknowledgment is prepared. When the SCP is in a Broadcast state, broadcasts are piggybacked or an acknowledgment is prepared.

One advantage of the present invention is that applications and data cached on each mobile device 110 are customized in accordance with each particular user's

operation history. That is, less frequently used applications and data cached on each mobile device 110 are continuously replaced by more frequently used applications and data. Thus, pre-loaded applications/data on a mobile device 110 can eventually be adapted to individual users.

The foregoing examples illustrate certain exemplary embodiments of the invention from which other embodiments, variations, and modifications will be apparent to those skilled in the art. The invention should therefore not be limited to the particular embodiments discussed above, but rather is defined by the claims.